Programming Engineering

Course 5 – 22 March 2017

adiftene@info.uaic.ro

Content

- From previous courses…
- Forward and Reverse Engineering
- GRASP
  - Information Expert
  - Creator
  - Low coupling
  - High cohesion
  - Controller

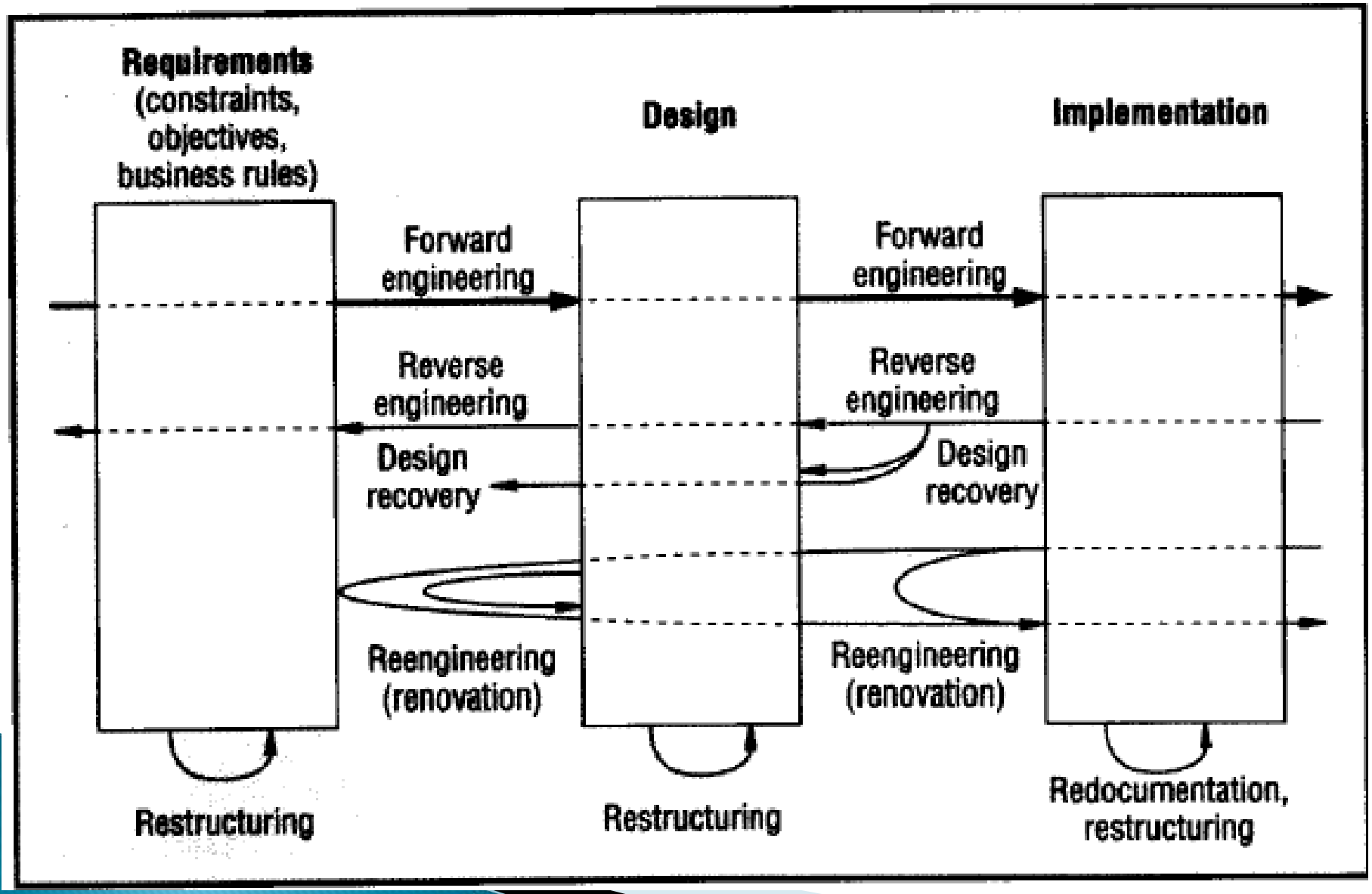
Attention

- The deadline for selecting the subject of the project is the 7th week
- After that: documentation, understanding, knowledge transfer, use case diagrams, class diagrams, implementation, unit testing, etc.
- Working at the project will start in the 7th week and it will end in the 14th week
-
- In week 8 will not be classes…

RE

- Why do we need modelling?
- How can we model a project?
- SCRUM – roles, values, artifacts, events, rules

Forward and Reverse Engineering



Forward Engineering

- A traditional process of moving from high-level abstractions and logical to the implementation-independent designs to the physical implementation of a system

- FE follows a sequence of going **from requirements through designing its implementation**

Reverse Engineering

- **Reverse engineering** (RE) is the process of discovering the technological principles of a device, object or system through analysis of its structure, function and operation
- *To try to make a new device or program that does the same thing without copying anything from the original*
- Reverse engineering has its origins in the analysis of hardware for commercial or military advantage

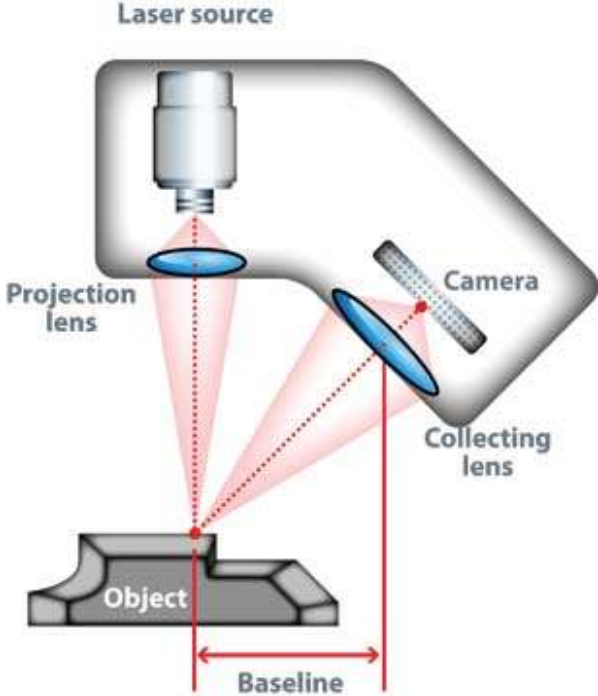
RE Motivation

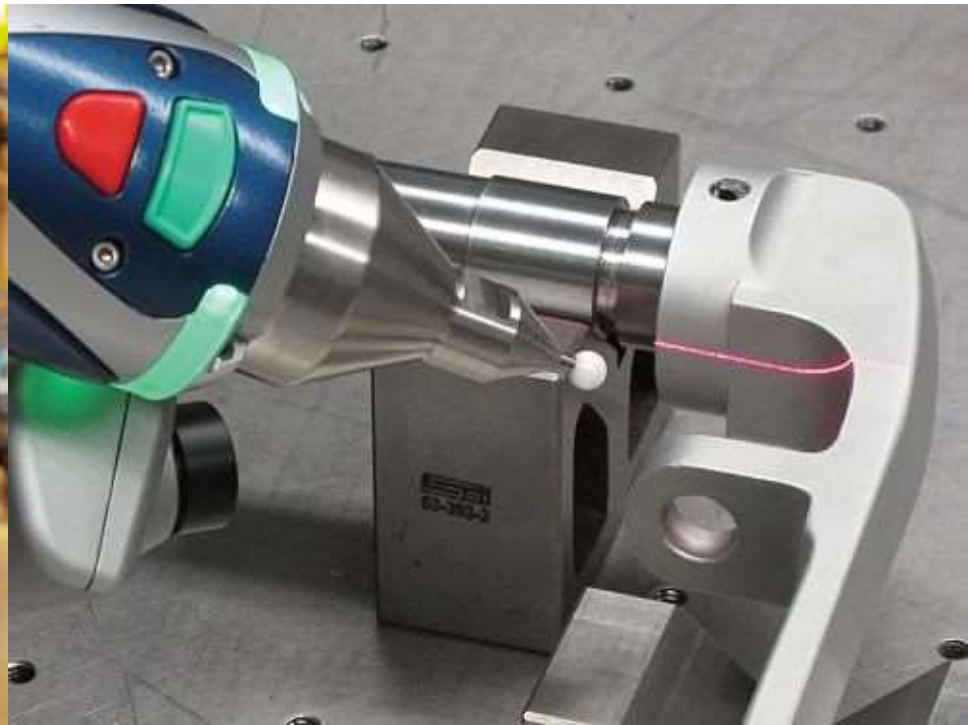
- Interoperability
- Lost documentation
- Product analysis
- Security auditing
- Removal of copy protection, circumvention of access restrictions
- Creation of unlicensed/unapproved duplicates
- Academic/learning purposes
- Curiosity
- Competitive technical intelligence (understand what your competitor is actually doing versus what they say they are doing)
- Learning: Learn from others mistakes

Types of RE

- RE1: Reverse engineering of mechanical devices
- RE2: Reverse engineering of integrated circuits/smart cards
- RE3: Reverse engineering for military applications
- RE4: Reverse engineering of software

RE1: 3D laser scanners





RE1: 3D Modeling Services CAD

Autodesk Design Review - C:\My Documents\Events\AU2006\Classes\AU_DataSets\Manufacturing\Shaver Complete.dwf

File Edit View Tools Help

Contents

[5] Shaver Complete S... [6] Drawing - Shaver Com... [7] PARTS LIST [8] Shaver Complete.iam [9] Shaver Complete S... [10] Shaver Complete S...

Markups
Properties
Layers
Cross Sections
Model

- Shaver Complete.iam
  - Housing Complete Left:1
  - Housing Complete Right:1
  - Cover:1
  - NOR-A-005:1
  - Switch:1
  - Shaver Complete.Harness
  - NOR-P-001:1
  - NOR-P-001:2
  - NOR-P-001:3
  - NOR-A-006:1
  - NOR-P-011:1
  - ISO 7045 Z M1.6 x 8 - 4.8 - Z:2
  - ISO 7045 Z M1.6 x 8 - 4.8 - Z:1

Views

Shaver Complete.iam 4 of 4

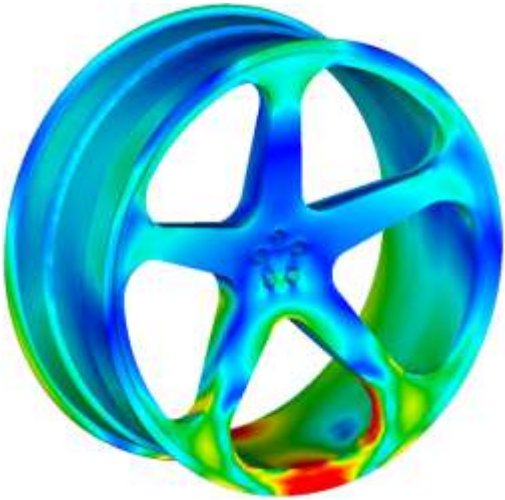
FRONT RIGHT

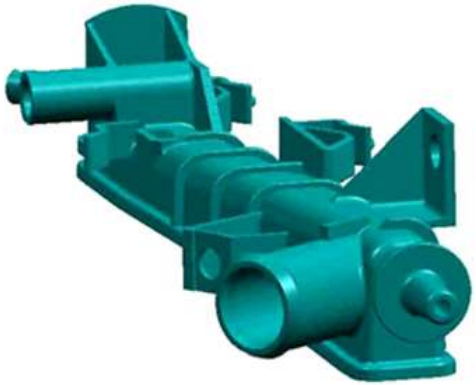
CENTER ZOOM REWIND ORBIT

Orbit around Center

1 of 6

ITEM	QTY	P/N	TITLE	DESCRIPTION
1	1	ADSK-1101010	ELECTRONICS	PCB
3	1	ADSK-1101041	ELECTRONICS	MOTOR
4	1	ADSK-1101015	ELECTRONICS	FRONT COVER
5	3	ADSK-1101031	FORMED PART	BLADE HEAD GUARD
2	1	ADSK-1101077	ELECTRONICS	HARNESS
6	1	ADSK-1101076	MOLDED PART	HOUSING LEFT
7	2	ADSK-253001	FASTENER	PAN HEAD



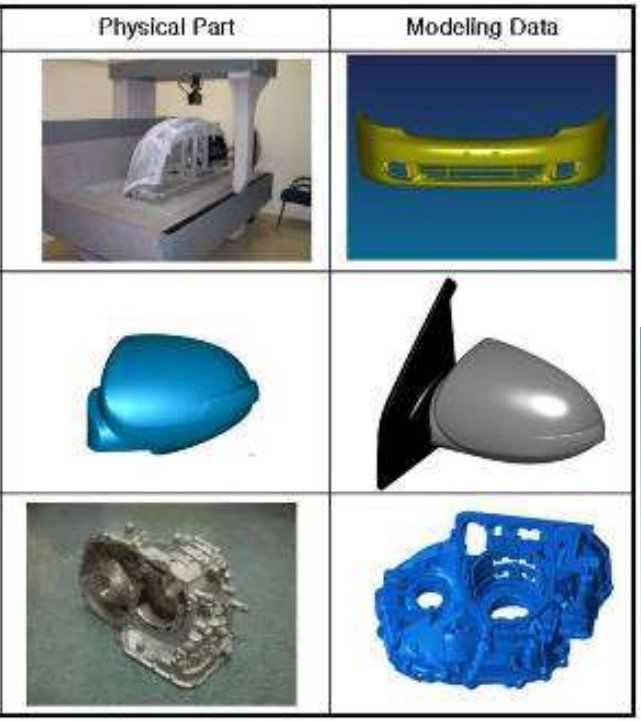


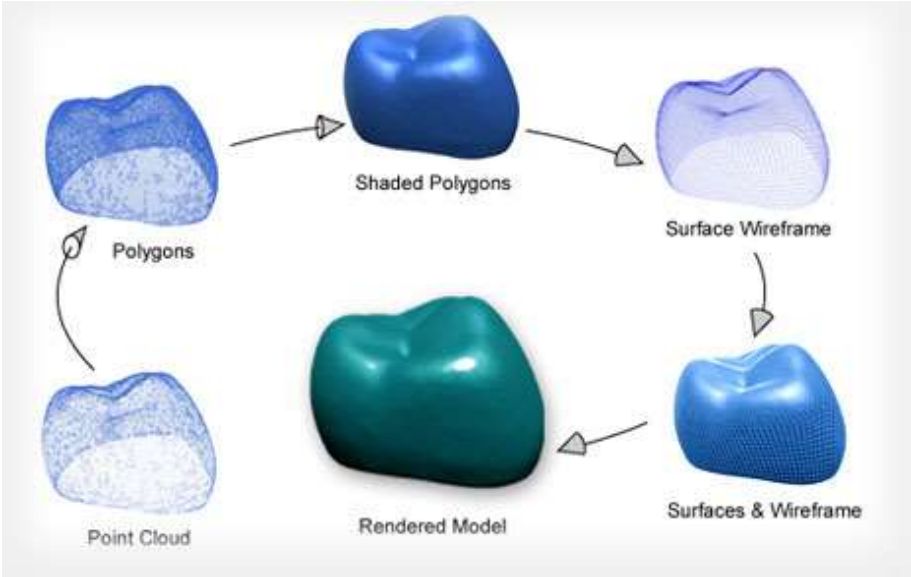
RE1: 3D Printing Services

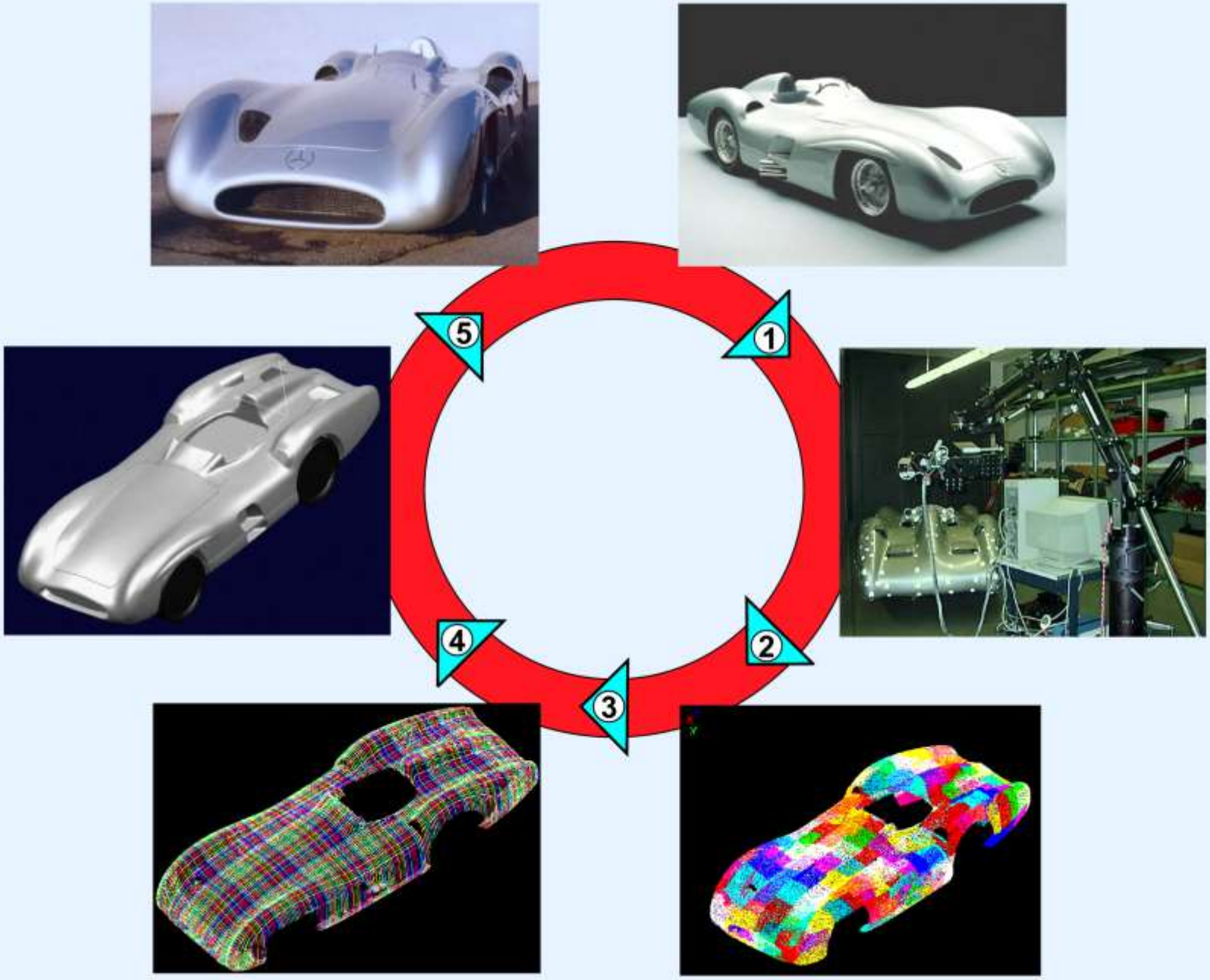
- Rapid prototyping

- FullCure materials

RE1: Areas


Physical Part
Modeling Data


Polygons
Shaded Polygons
Surface Wireframe
Surfaces & Wireframe
Rendered Model
Point Cloud


1
2
3
4
5

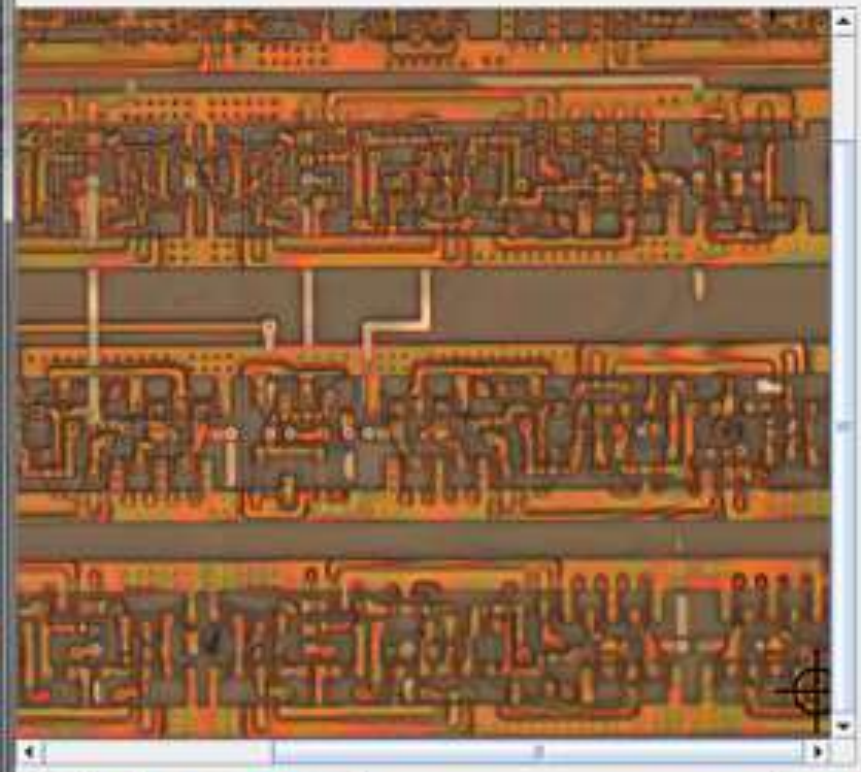
Reverse engineering of integrated circuits/smart cards

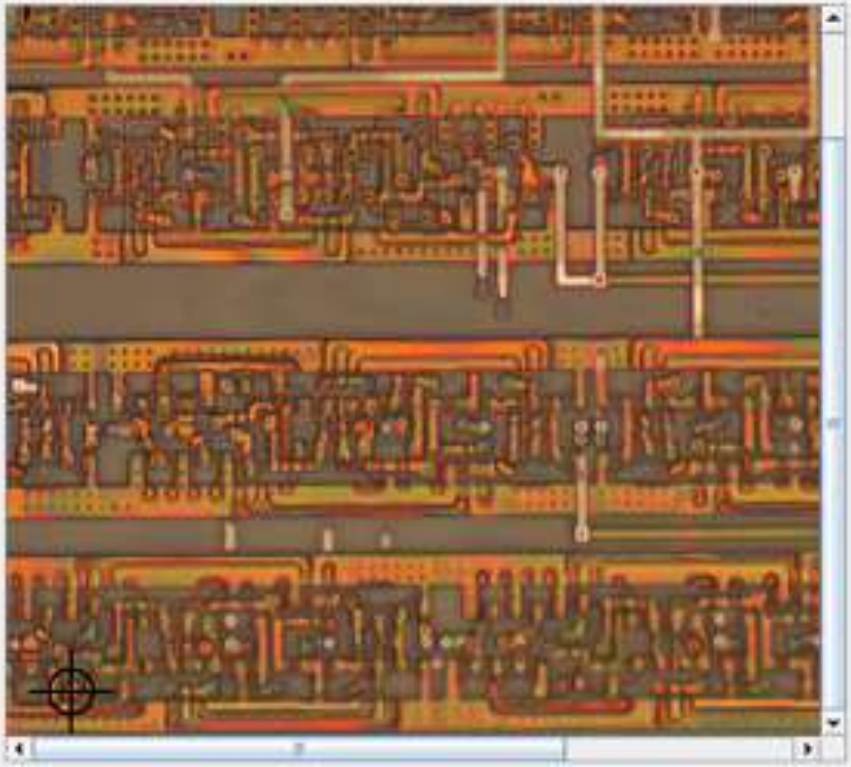
- RE is an invasive and destructive form of analyzing a smart card
- The attacker grinds away layer by layer of the smart card and takes pictures with an electron microscope
- Engineers employ sensors to detect and prevent this attack

Image Stitching Tool

3. Schritt: Setzen der Kontrollpunkte

D:\... TFH DATEIEN ...\xx Praktika\CRE\aChipBilder\iz...





654 - 63

680 - 484

12 - 64

38 - 486

Kontrollpunkt speichern + Weiter

degate -- [/home/martin/Development/degate3/2011-02-06_0045_degate*] [1/3]

Project View Tools Layer Logic Gate Recognition Help

Logic gates

Short Name	#	Width	Height	Fill color	Frame color	Description
01-FF	58	272	134			D-Q-FlipFlop
02-FF	11	343	134			D-Q-FlipFlop with rst
03-FF	17	343	133			D-Q-FlipFlop with rts
04-BUF	5	226	128			
05-3XOR	13	249	133			
06: 2-XNOR	12	133	133			

Edit Add Remove

Close

Edit gate

Entity Behaviour Layout

Short name: 02-FF

Description: D-Q-FlipFlop with rst

Logic Class: flipflop (generic)

Ports:

Port ID	Port Name	Port Description	In	Out
2384	!Q	!Q	☐	☑
2380	Q	Q	☐	☑
2388	clk	clk	☑	☐
2386	D	D	☑	☐
2382	rst	rst	☑	☐

Add Remove

Fill color: Reset Color

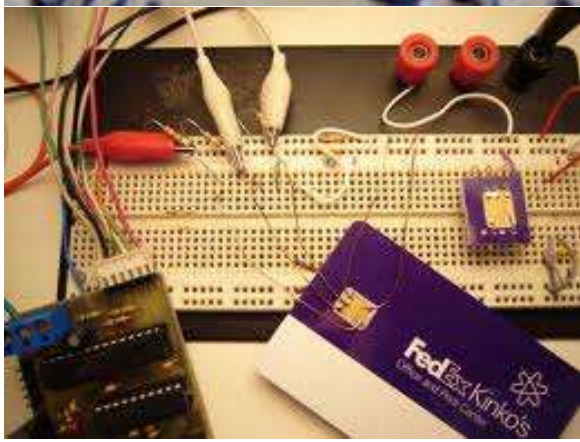
Frame color: Reset Color

Cancel OK

Autosaving project data ... done.

RE2: Smart cards

- Satellite TV
- Security card
- Phone card
- Ticket card
- Bank card



Reverse engineering for military applications

- Reverse engineering is often used by militaries in order to copy other nations' technologies, devices or information that have been obtained by regular troops in the fields or by intelligence operations

- It was often used during the Second World War and the Cold War

- Well-known examples from WWII and later include: rocket, missile, bombers, China has reversed many examples of US and Russian hardware, from fighter aircraft to missiles and HMMWV cars

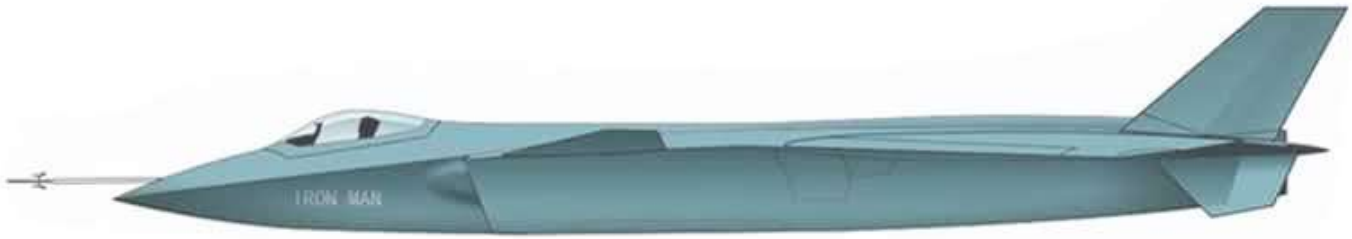
RE3: Aircrafts

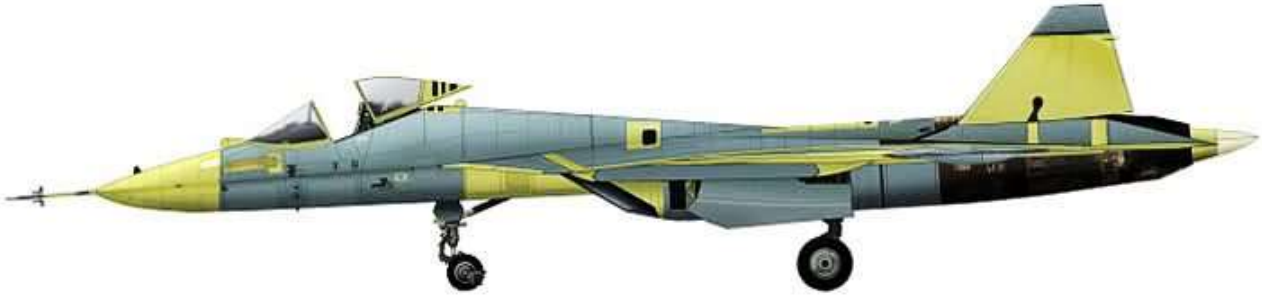
- US – B-29

URSS – Tupolev Tu-4

RE3: Aircrafts (2)

- Chinese J-20, Black Eagle US F-22, Russian Sukhoi T-50



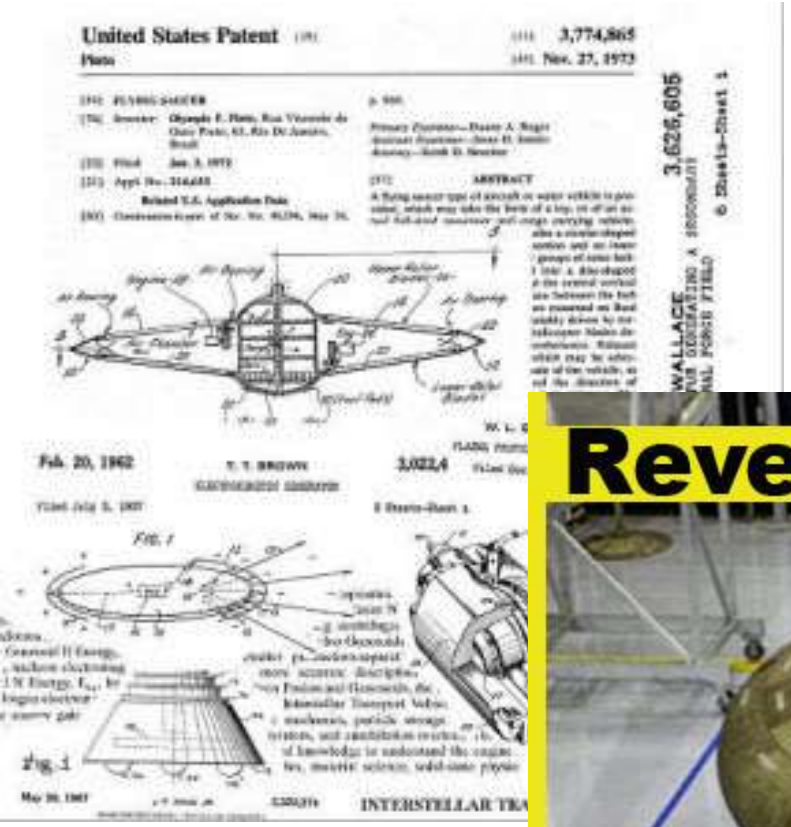


RE3: Rockets

- US –AIM-9 Sidewinder Soviet – Vympel K-13

RE3: Submarine

RE3: UFOs


United States Patent
3,774,865
Nov. 27, 1973
Feb. 20, 1962
INTERSTELLAR TRA


Reverse Engineering a UFO
by Robert Klein

Reverse engineering of software

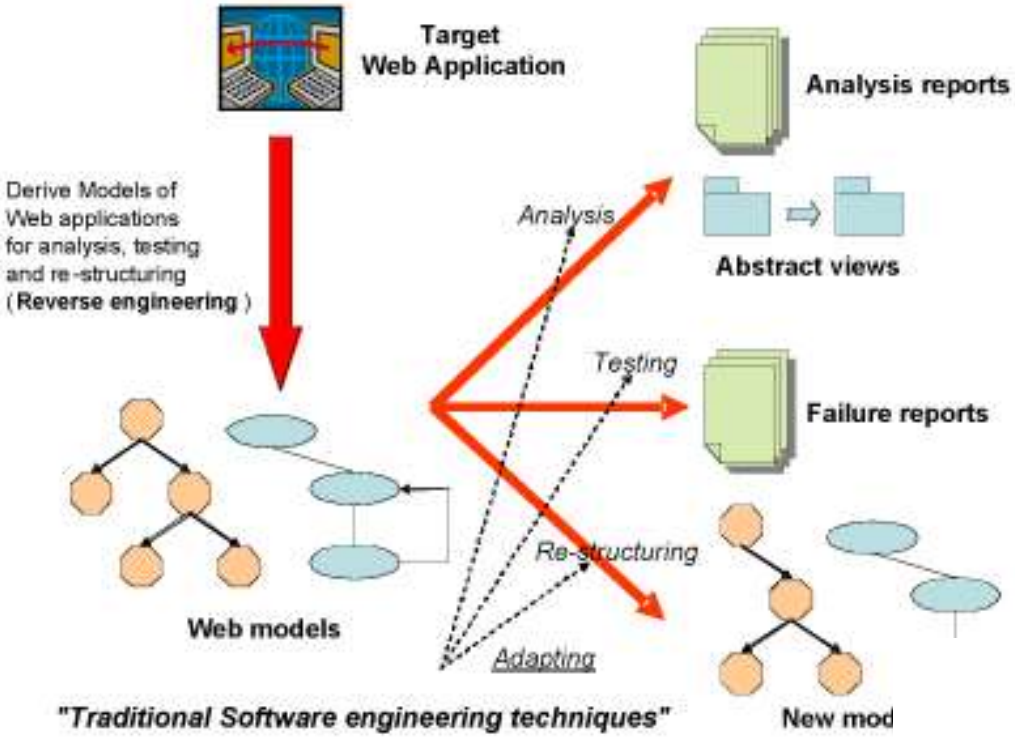
- Reverse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction

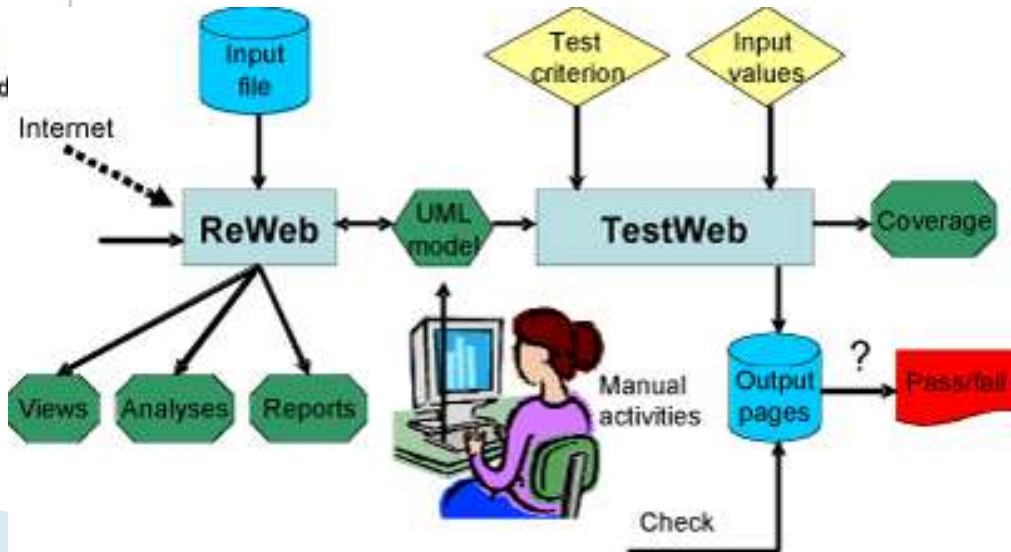
- In practice, two main types of RE emerge:
  - **Source code is available** (but it is poorly documented)
  - **There is no source code available for the software**

- **Black box testing** in software engineering has a lot in common with reverse engineering

RE4: Smart phones

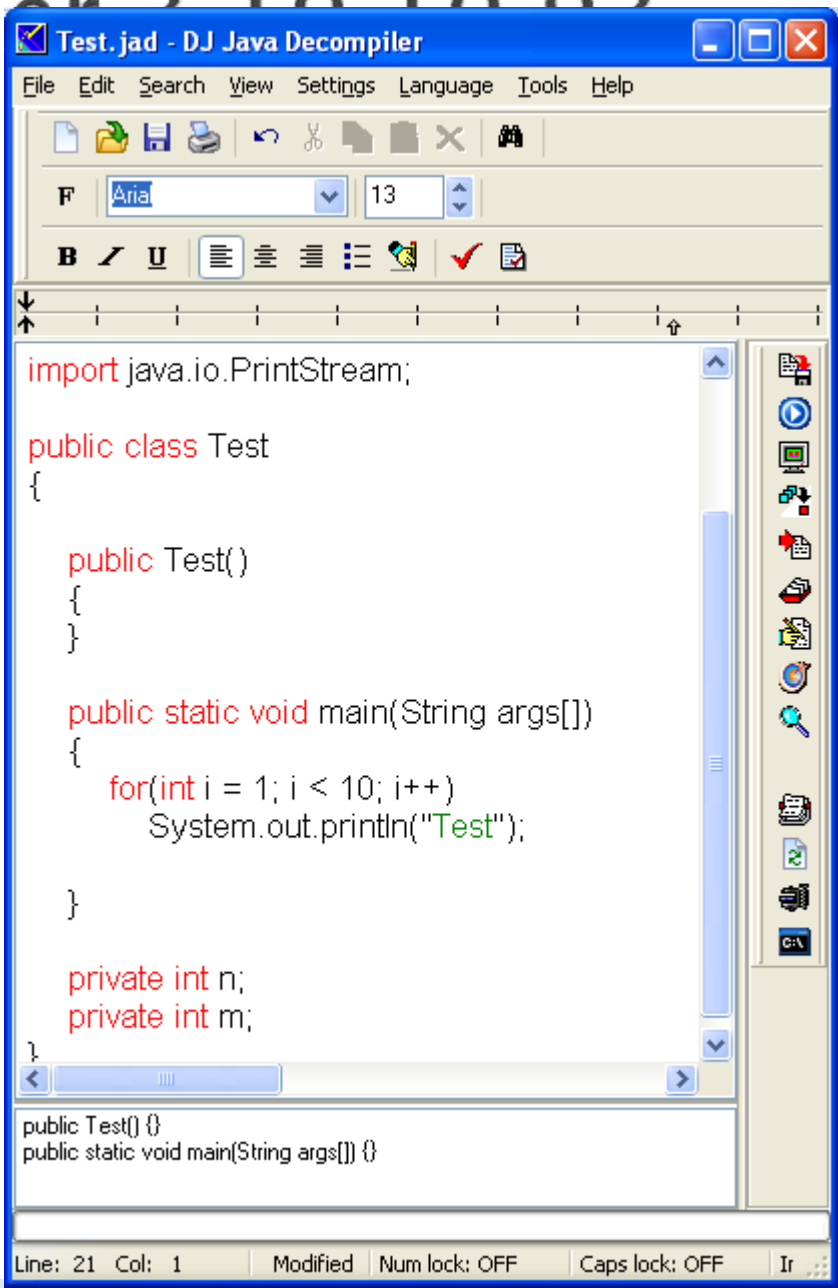
RE4 of Web Applications





RE4: DJ Java Decompiler 3.10.10.93

```
public class Test
{
   private int n;
   private int m;

   public static void main(String
   args[])
   {
         for(int i=1;i<10;i++)
         System.out.println("Test");
   }
}
```



RE4: JAD

- Link: http://www.steike.com/code/java-reverse-engineering/
- jad.exe FileName.class => FileName.jad

D:\Work\Ady\Scoala\2009-2010\JP\Cursuri\04\Reverse\Initial\Student.java - Notepad++

Student.java

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package test1;

/**
 *
 * @author Adrian
 */
public class Student extends Person{
    private int year;
    private int group;

    public Student(String name, String address,int year, int gro
    {
        super(name,address);
        this.group = group;
        this.year = year;
    }
}
```

371 chars 394 bytes 23 lines | Ln : 23 Col : 1 Sel : 0 (0 bytes) in 0 ranges | UNIX | ANSI

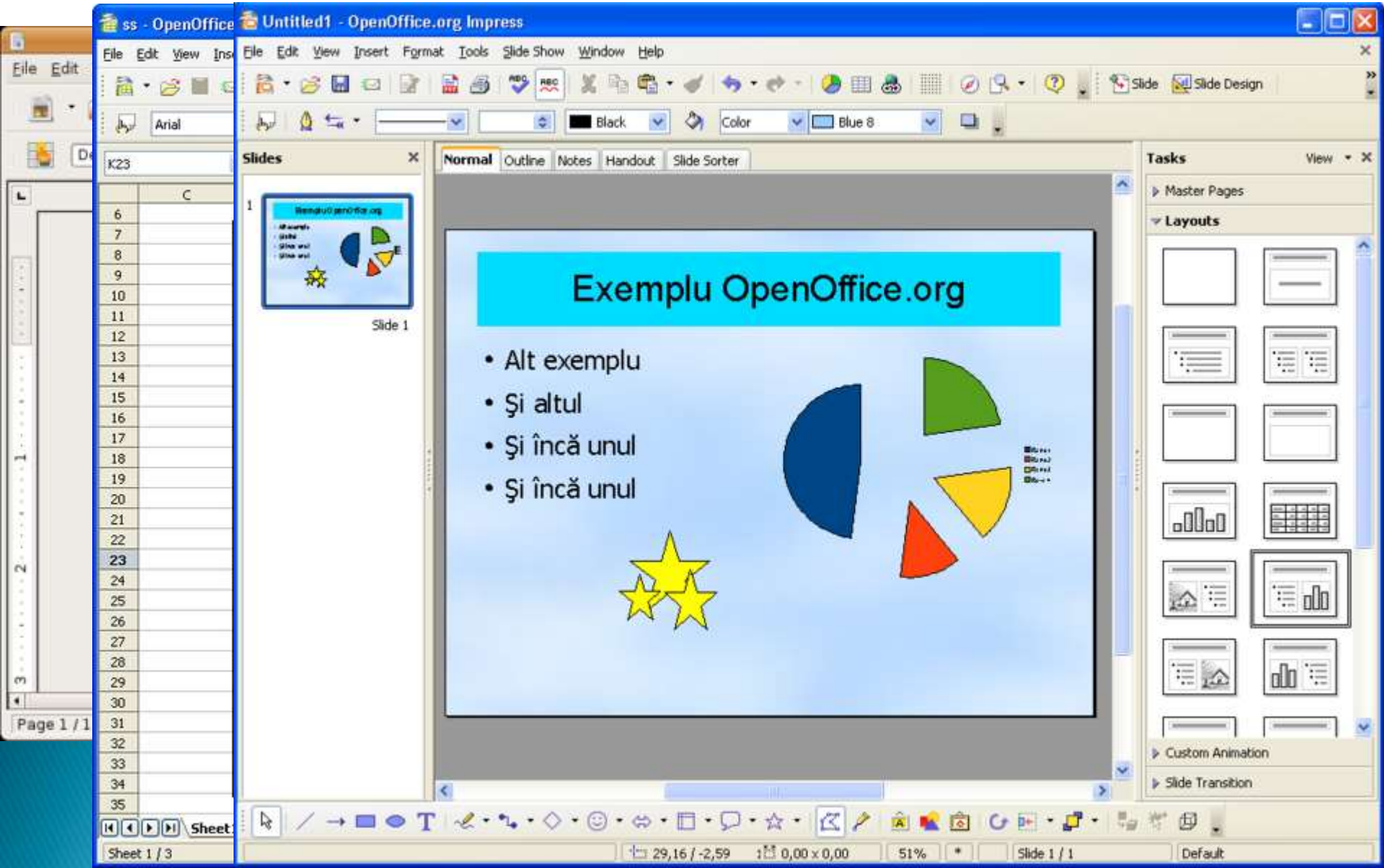
D:\Work\Ady\Scoala\2009-2010\JP\Cursuri\04\Reverse\Student.jad - Notepad++

Student.jad

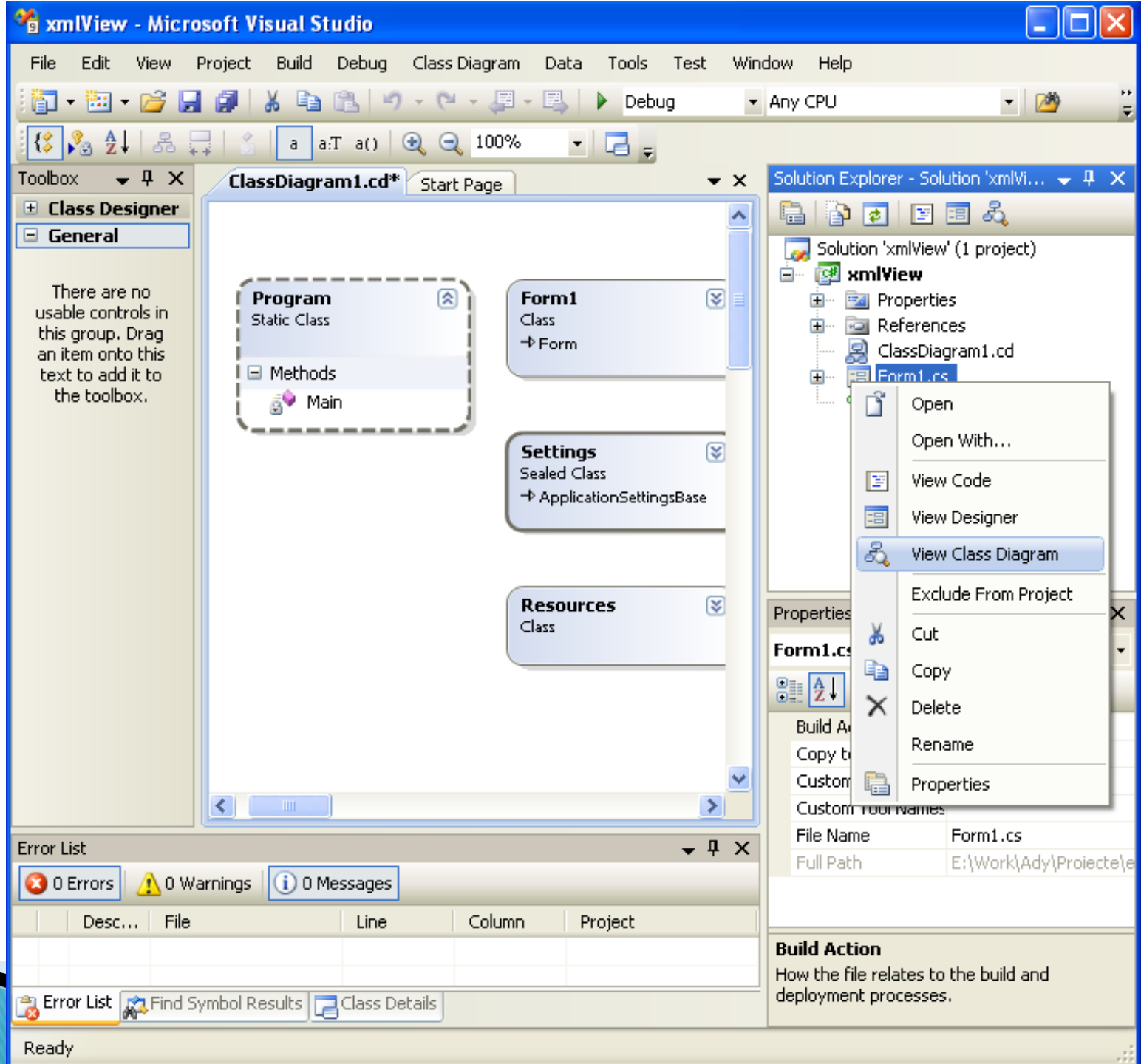
```java
// Decompiled by Jad v1.5.8f. Copyright 2001 Pavel Kouznetsov.
// Jad home page: http://www.kpdus.com/jad.html
// Decompiler options: packimports(3)
// Source File Name:   Student.java

package test1;


// Referenced classes of package test1:
//            Person

public class Student extends Person
{

    public Student(String name, String address, int year, int group)
    {
        super(name, address);
        this.group = group;
        this.year = year;
    }

    private int year;
    private int group;
}
```

Java : 492 chars 542 bytes 25 lines | Ln : 1 Col : 1 Sel : 0 (0 bytes) in 0 ranges | Dos\Windo | ANSI | INS

RE4: Open Office

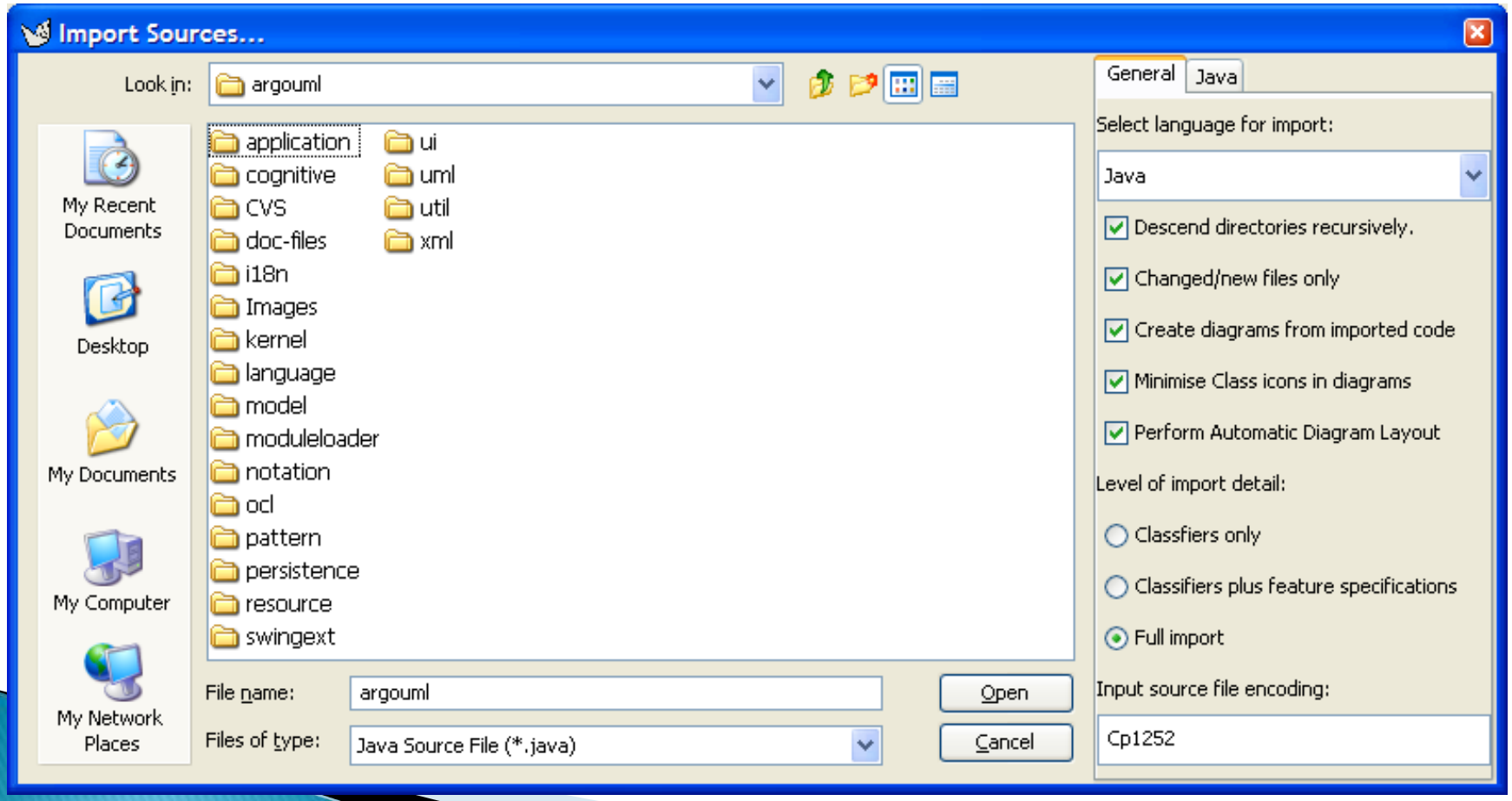


C#

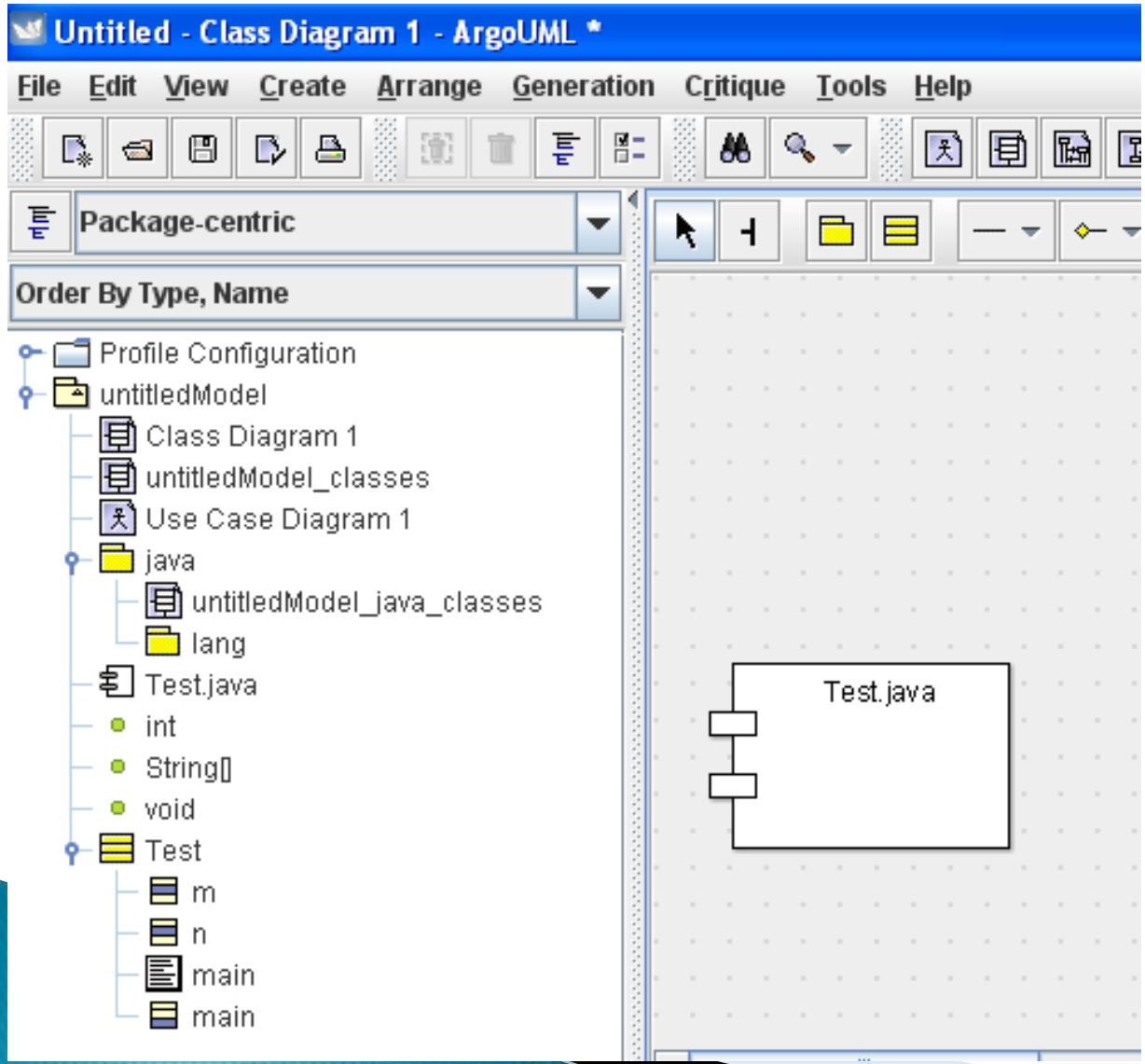

xmlView - Microsoft Visual Studio
File Edit View Project Build Debug Class Diagram Data Tools Test Window Help
Debug
Any CPU
100%
Toolbox
Class Designer
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
ClassDiagram1.cd*
Start Page
Program
Static Class
Methods
Main
Form1
Class
Form
Settings
Sealed Class
ApplicationSettingsBase
Resources
Class
Solution Explorer - Solution 'xmlVi...
Solution 'xmlView' (1 project)
xmlView
Properties
References
ClassDiagram1.cd
Form1.cs
Open
Open With...
View Code
View Designer
View Class Diagram
Exclude From Project
Cut
Copy
Delete
Rename
Properties
Form1.cs
File Name
Form1.cs
Full Path
Build Action
How the file relates to the build and deployment processes.
Error List
0 Errors
0 Warnings
0 Messages
Desc...
File
Line
Column
Project
Error List
Find Symbol Results
Class Details
Ready

RE in ArgoUML

- File –> **Import Sources…**



For previous example…



GRASP

- GRASP = General Responsibility Assignement Software Patterns (Principles)
- Described by Craig Larman in *Applying UML and Patterns. An Introduction to Object Oriented Analysis and Design*
- **It helps us assign responsibilities for classes and objects in the most elegant way possible**
- Examples of principles used in GRASP: *Information Expert* (or Expert), *Creator*, *High Cohesion*, *Low Couplig*, *Controller* Polymorphism, Pure Fabrication, Indirection, Protected Variations

What responsibilities?

- To do:
  - To do something himself, as well as creating an object or make a calculation
  - Initialization of an action in other objects
  - Controlling and coordinating other objects

- To know:
  - Private attributes
  - Its own objects
  - The things you can do or those you can call

Pattern

- Also known as: template, model
- It is a general solution to a common problem
- Each pattern has a catchy and meaningful name (eg. Composite, observer, iterator, singleton, etc.)

Information Expert 1

- **Problem**: to which classes a given behavior (operation) has to be assigned?

- **A better allocation of operations leads to systems that are:**
  - Easy to understand
  - More easily extended
  - Reusable
  - More robust

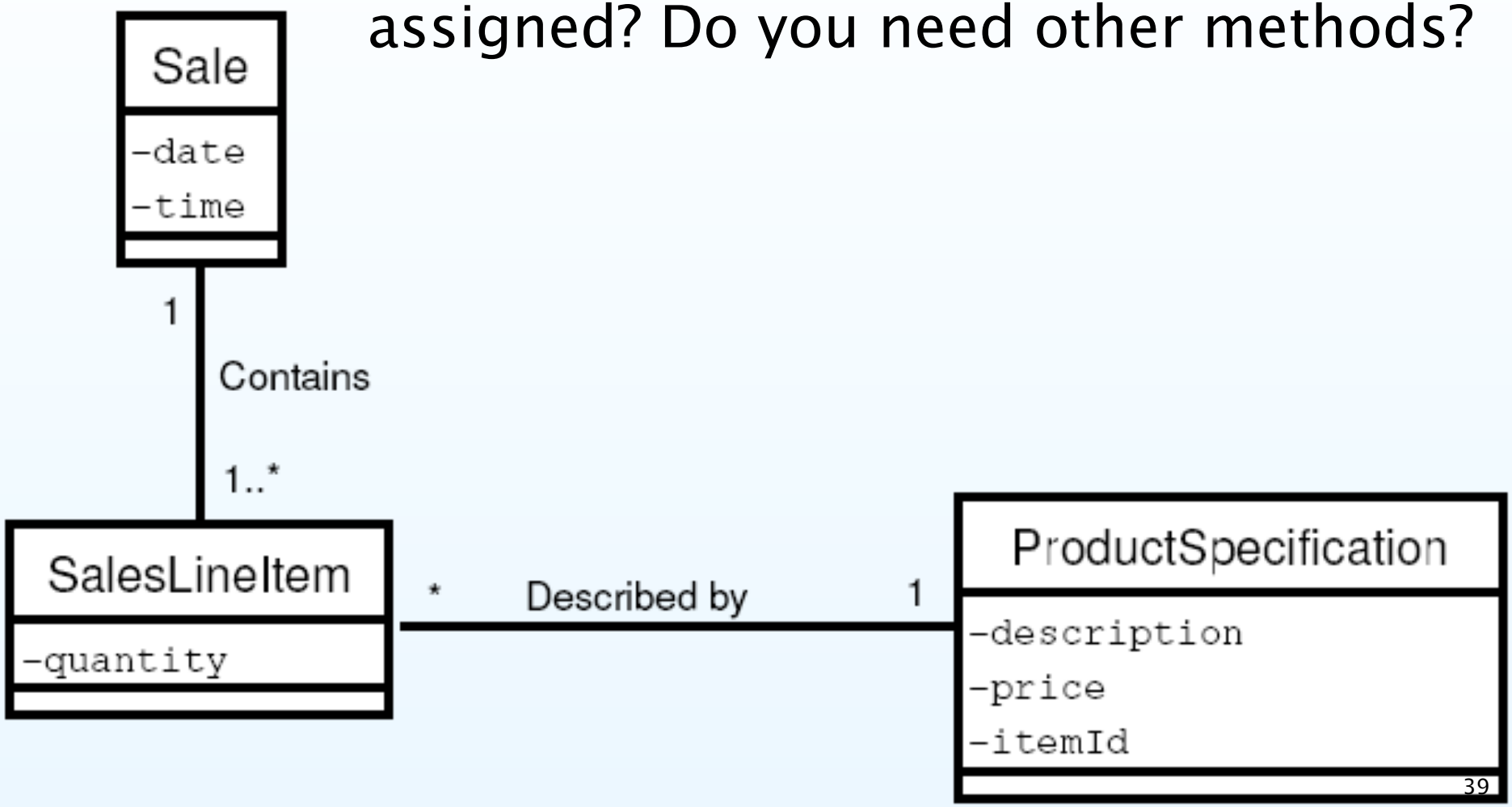
Information Expert 2

- **Solution**:
- Start assigning responsibilities to a class that has the information necessary to fulfill those responsibilities

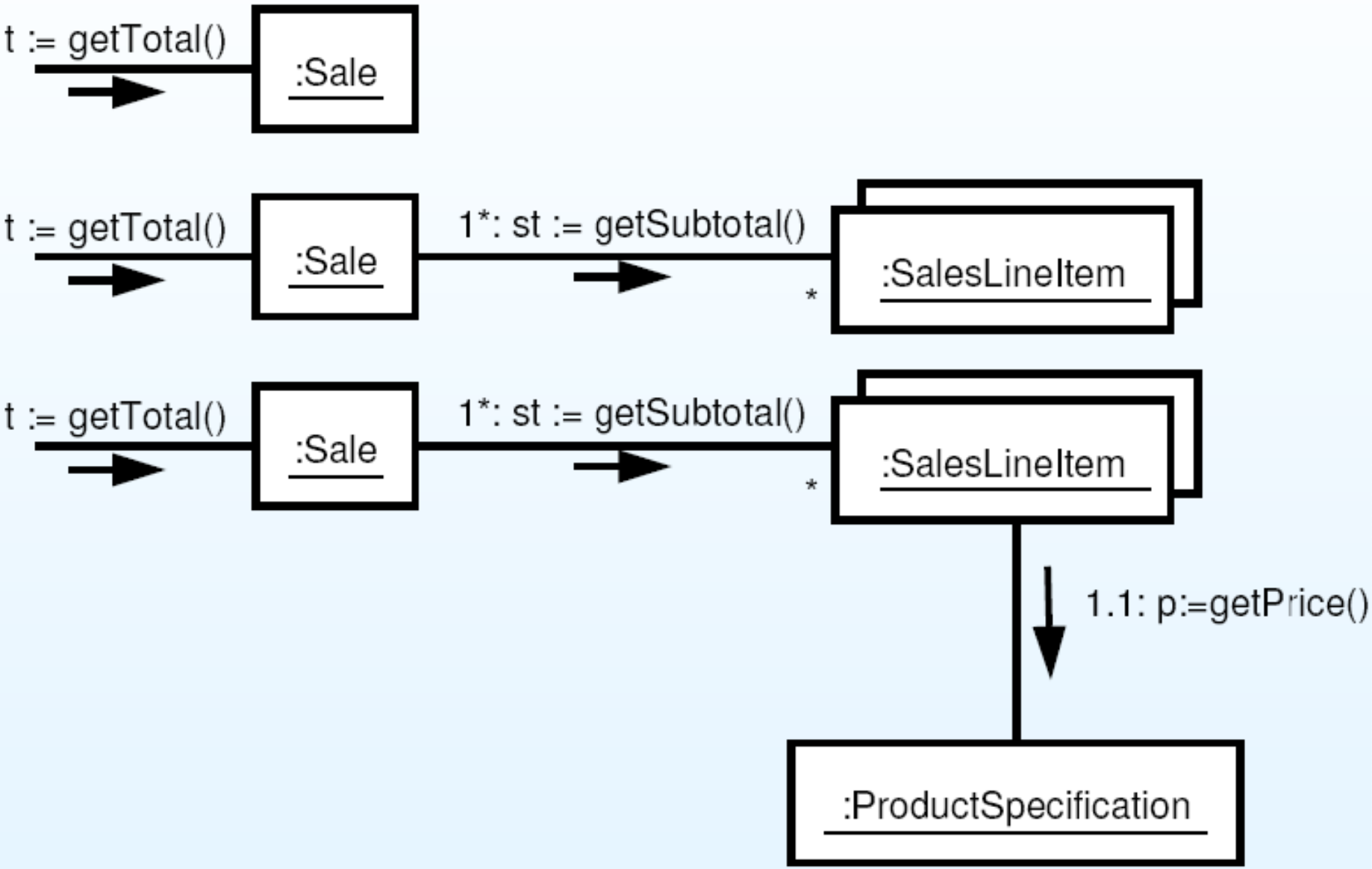
- **Recommendation:**
- start assigning the responsibilities clearly highlighting them

Example 1

- To which classes should method getTotal() be assigned? Do you need other methods?

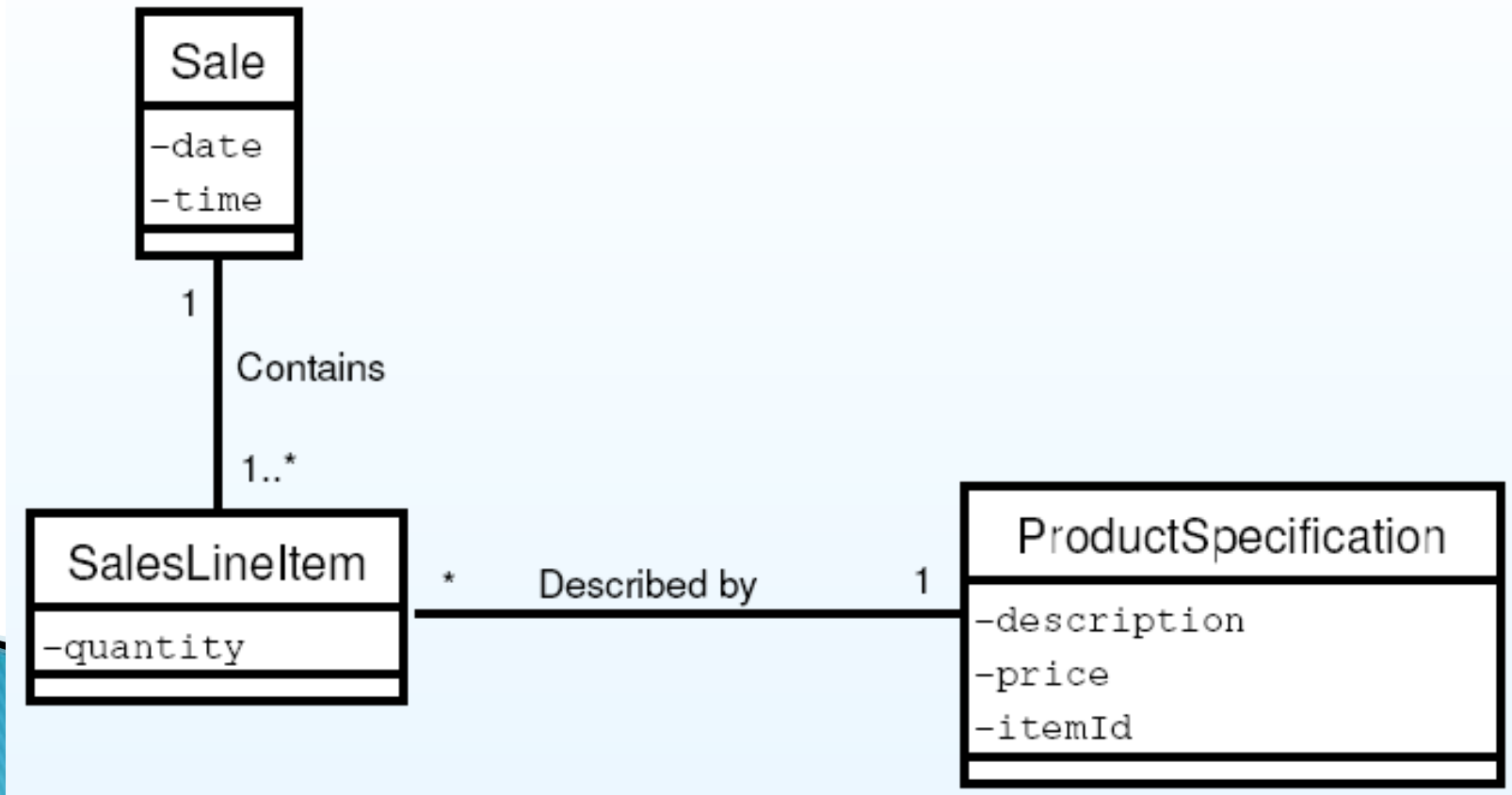


Example 2

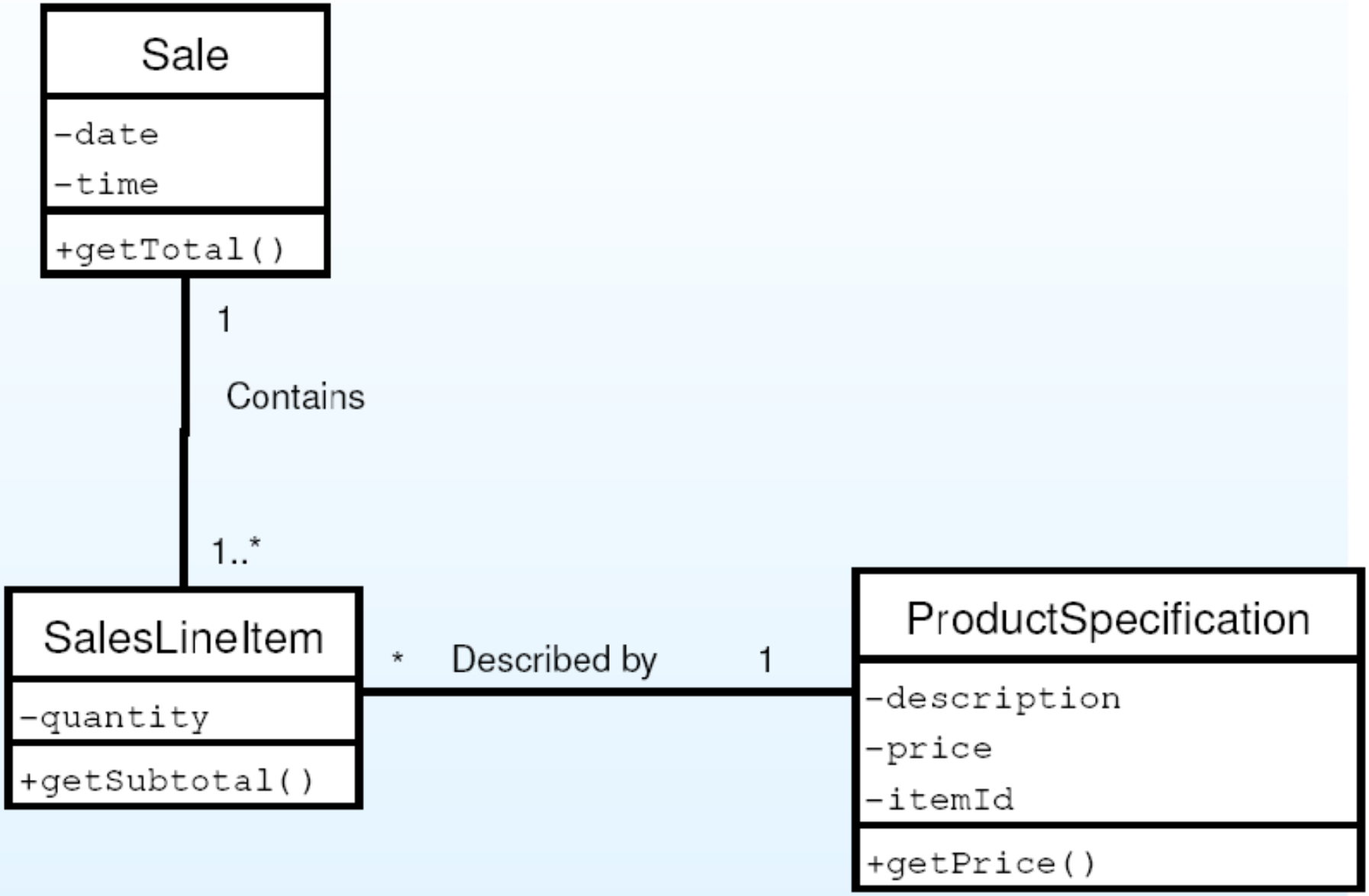

t := getTotal()
:Sale
t := getTotal()
:Sale
1*: st := getSubtotal()
*
:SalesLineItem
t := getTotal()
:Sale
1*: st := getSubtotal()
*
:SalesLineItem
1.1: p:=getPrice()
:ProductSpecification

Possible solution 1

Class	Responsibilities
Sale	to know the total value purchase
SalesLineItem	to know the subtotal for an item
ProductSpecification	to know the price of the product



Possible solution 2

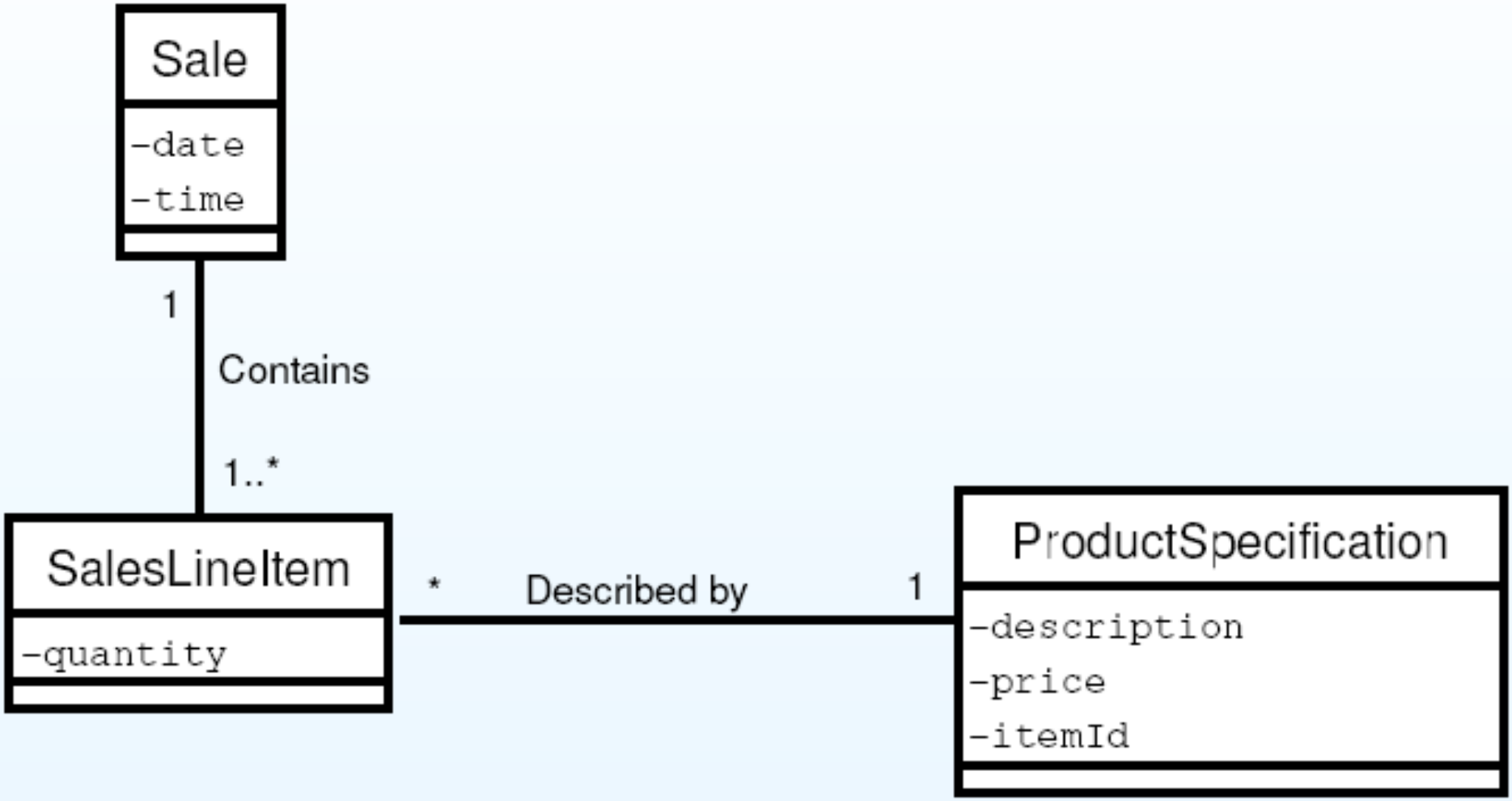


Creator 1

- **Problem**: Who should be responsible for creating an instance of a class?
- **Solution**: Assign class B the responsibility to create instances of class A only if at least one of the following is true:
  - B aggregates objects of type A
  - B contains objects of type A
  - B uses objects of type A
  - B has initialization data to be transmitted to instantiate an object of type A (B is therefore an expert in terms of creating objects of type A)
- **Factory pattern** is a more complex variant

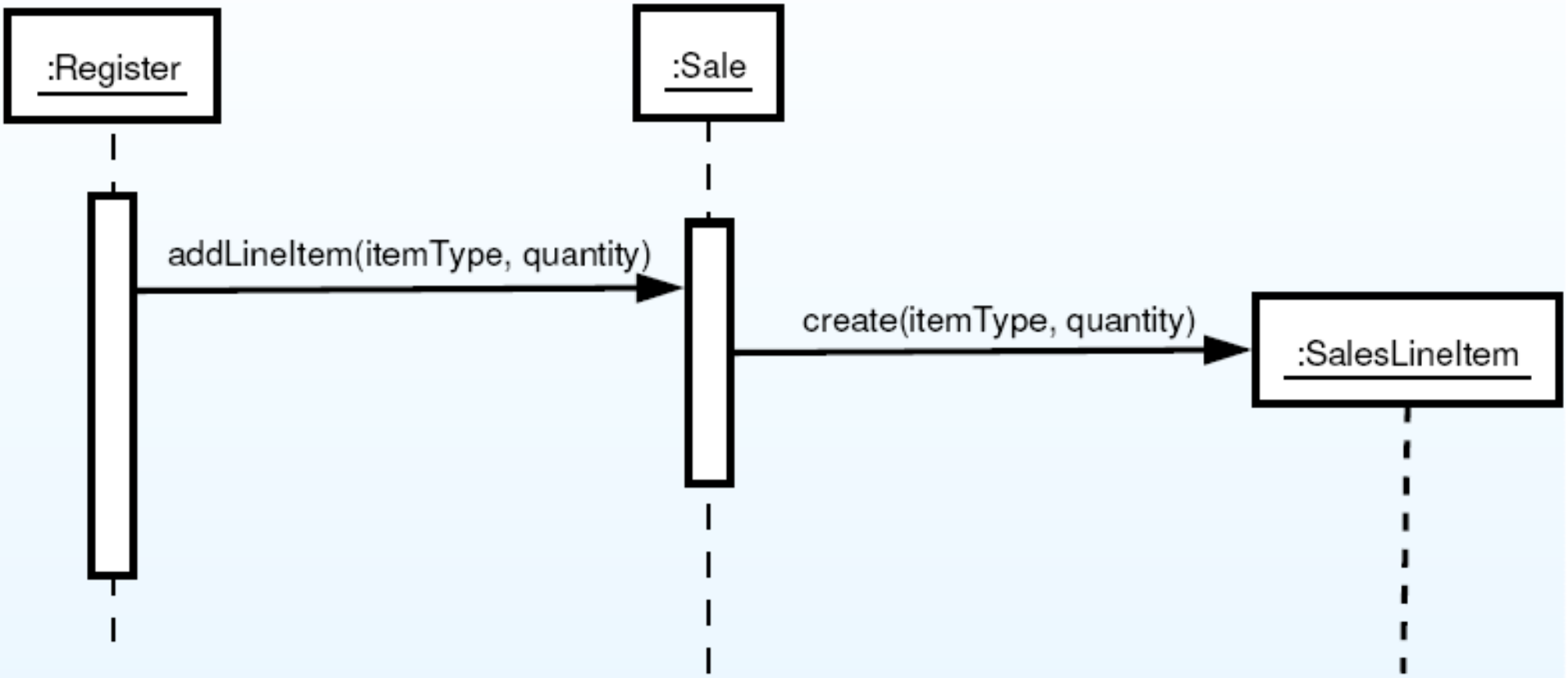
Creator 2

- Who is responsible for creating an instance of the class SalesLineItem?



Creator 3

- Because Sale contains (aggregates) instance type SalesLineItem, it is a good candidate to be assigned the responsibility for creating these instances



Low coupling

- The coupling is a measure of the degree of dependence of a class on other classes
- Types of Dependence:
  - It is connected to
  - It has knowledge about
  - It is based on
- A class that has low coupling (reduced) does not depend on "many" other classes; where "many" depends on context
- A class that has high coupling depends on many other classes

Coupling 2

- Problems caused by coupling:
  - changes in the related classes force local changes
  - difficult to understand classes in isolation (out of context)
  - hard to reuse classes because their use requires the presence of dependent classes

Coupling 3

- Common forms of coupling from class A to class B are:
  - A has an attribute of type B
  - An instance of the class A calls a service offered by an object of type B
  - A has a method that references B (parameter, local object, the object returned)
  - A is a subclass (direct or indirect) of B
  - B is an interface, and A implements this interface

Law of Demeter

- *Don't talk to strangers*

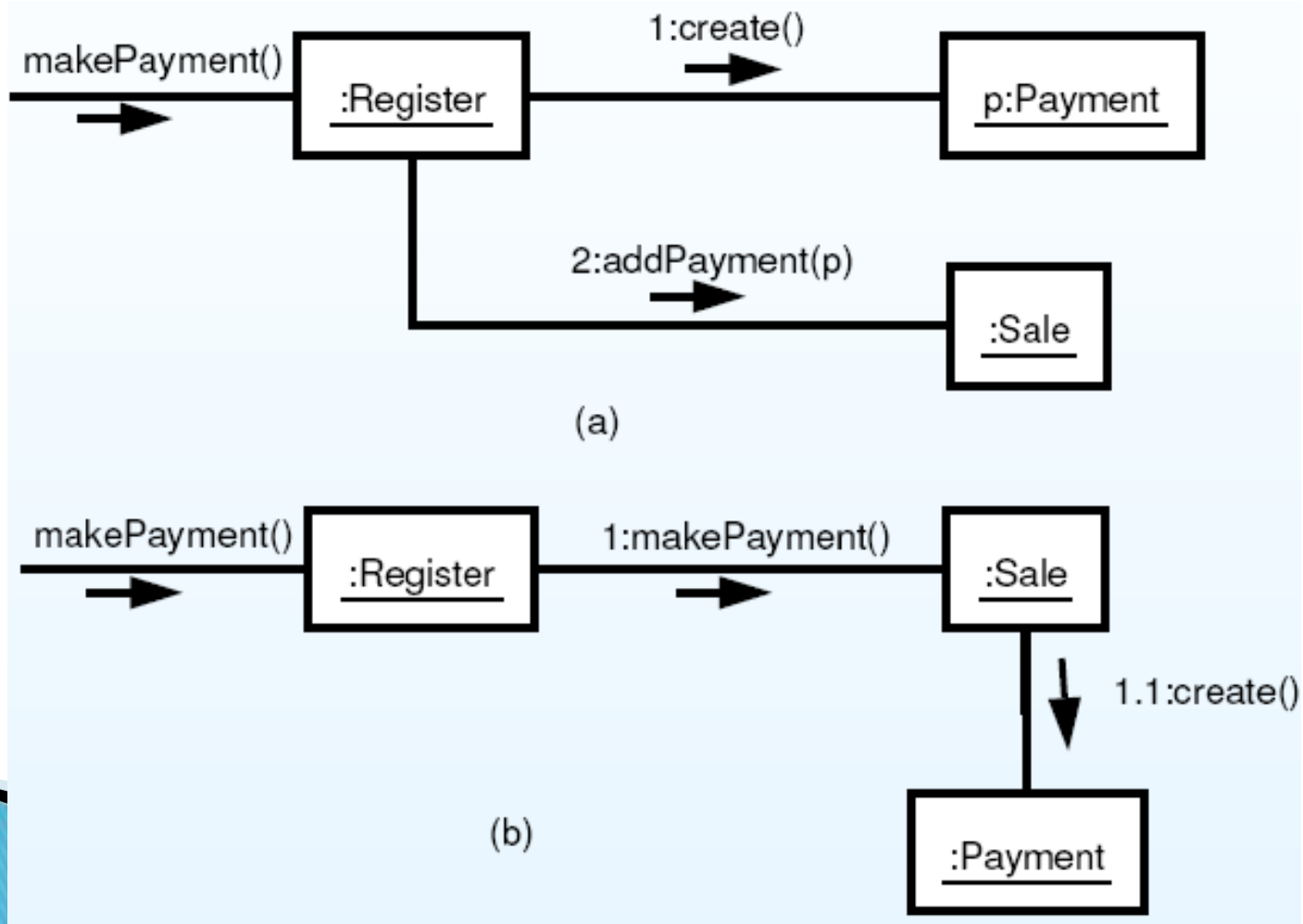
- Any method of an object should call only methods belonging to:
  - himself
  - any parameter of the method
  - any object that it created
  - any objects that it contains

The visualization of couplings

- Class Diagram
- Collaboration Diagram

Example 1

- There are links between all classes
- It eliminates coupling between Register and Payment



High Cohesion

- **Cohesion** is a measure of how strong the responsibilities of a class are focused
- A class whose responsibilities are very closely linked and which is not very much has a **great cohesion**
- A class that does many things that are not related to each other or that does too many things has a **low (weak) cohesion**

Cohesion

- Problems caused by a weak cohesion:
  - hard to understand
  - hard to reused
  - hard to maintain
  - delicate; such classes are always subject to change

Cohesion and coupling

- Are old principles in software design
- Promote a modular design
- Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules

Controller 1

- **Problem**: Who is responsible for dealing with an event generated by an actor?
- These events are associated with operations of the system
- A Controller is an object that does not belong to the graphical interface, which is responsible for receiving or managing an event
- A controller defines a method for a corresponding operation of the system

Controller 2

- **Solution:** assign the responsibility for receiving or managing an event to a class representing one of the following choices:
  - Represents the entire system or subsystem (facade controller)
  - It is a usage scenario in which the event occurs

Controller 3

- Normally a controller should delegate to other objects the work that is to be done
- The controller coordinates and controls the activity, but it does not do too many things himself
- A common mistake in the design of a controller is to assign it too many responsibilities (facade controller)

Conclusions

- Forward & Reverse Engineering

- GRASP
  - Information Expert
  - Creator
  - Low coupling
  - High cohesion
  - Controller

Bibliography

- Reverse Engineering and Design Discovery: A Taxonomy, Chikofsky, E.J. and Cross, J., January, 1990
- Craig Larman. *Applying UML and Patterns. An Introduction to Object Oriented Analysis and Design*
- Ovidiu Gheorghieș, Course 6 IP

Links (RE)

- **DJ Java Decompiler 3.10.10.93:** http://www.softpedia.com/progDownload/DJ-Java-Decompiler-Download-13481.html
- **Open Office**: http://ro.wikipedia.org/wiki/OpenOffice.org
- **UML Reverse Engineering for Existing Java, C# , and Visual Basic .NET Code: http://www.altova.com/umodel/uml-reverse-engineering.html**
- **Reverse Engineering**: http://en.wikipedia.org/wiki/Reverse_engineering
- **PROTO 3000 3D Engineering Solutions:** http://www.proto3000.com/services.aspx
- HAR2009: http://www.degate.org/HAR2009/
- Degate: http://www.degate.org/screenshots/
- Inteligent: http://www.intelligentrd.com/
- Smartphones RE: http://www.cytraxsolutions.com/2011/01/smartphones-security-and-reverse.html

Links (GRASP)

- WebProjectManager: http://profs.info.uaic.ro/~adrianaa/uml/
- State Diagrams and Activity Diagrams: http://software.ucv.ro/~soimu_anca/itpm/Diagrame%20de%20Stare%20si%20Activitate.doc
- Deployment Diagram: http://en.wikipedia.org/wiki/Deployment_diagram http://www.agilemodeling.com/artifacts/deploymentDiagram.htm
- GRASP: http://en.wikipedia.org/wiki/GRASP_(Object_Oriented_Design)
- http://web.cs.wpi.edu/~gpollice/cs4233-a05/CourseNotes/maps/class4/GRASPpatterns.html
- Introduction to GRASP Patterns: http://faculty.inverhills.edu/dlevitt/CS%202000%20(FP)/GRASP%20Patterns.pdf